C++学习笔记8

2022年3月19日 20:55

- 关联容器容器类型
  - 类型map和mutimap定义在头文件map中，set和mutiset定义在头文件set中。无序容器则定义在unordered-map和unordered-set中

表 11.1：关联容器类型	
按关键字有序保存元素	
map	关联数组；保存关键字-值对
set	关键字即值，即只保存关键字的容器
multimap	关键字可重复出现的 map
multiset	关键字可重复出现的 set
无序集合	
unordered_map	用哈希函数组织的 map
unordered_set	用哈希函数组织的 set
unordered_multimap	哈希组织的 map；关键字可以重复出现
unordered_multiset	哈希组织的 set；关键字可以重复出现

- 关联容器
  - map类型的定义
    - map是关键词-值对的集合
    - map类型通常被称为关联数组。关联数组与正常数组类似，不同之处在于其下标不必是整数，我们通过一个关键词而不是位置来查找值
    - 定义一个map的时候，我们必须指定关键词和值的类型，关键词将作为下标，值将作为值
    - map的对应关系：key-value：key相当于数组 的下标，value相当于数组 的值
    - map的打印：对于map中每一个元素w.first-》下标，w。second-》数字
    - map会在你输入的时候按照关键词顺序给你进行排序（也就是说你必须有满足运算的操作符）
    - 对于map和mutimap，都是由pair构成的，map是一一对应的关系，不能重复，而mutimap是一对多，即可以保存相同关键词，不同值的内容
  - set的使用
    - set只保存值，关键字即值
  - pair类型
    - pair定义在utility
    - pair不是关联容器类型
    - pair保存两个数据成员，类似容器，这两个数据成员都是public 的
    - 我们使用点运算符访问成员
      w.first,w.second
    - pair的操作

□

表 11.2：pair 上的操作

`pair<T1, T2> p;`	p 是一个 pair，两个类型分别为 T1 和 T2 的成员都进行了值初始化（参见 3.3.1 节，第 88 页）
`pair<T1, T2> p(v1, v2)`	p 是一个成员类型为 T1 和 T2 的 pair；first 和 second 成员分别用 v1 和 v2 进行初始化
`pair<T1,T2>p = {v1,v2};`	等价于 p（v1,v2）
`make_pair(v1, v2)`	返回一个用 v1 和 v2 初始化的 pair。pair 的类型从 v1 和 v2 的类型推断出来
`p.first`	返回 p 的名为 first 的（公有）数据成员
`p.second`	返回 p 的名为 second 的（公有）数据成员
`p1 relop p2`	关系运算符（<、>、<=、>=）按字典序定义：例如，当 p1.first < p2.first 或!(p2.first < p1.first) && p1.second < p2.second 成立时，p1 < p2 为 true。关系运算利用元素的< 运算符来实现
`p1 == p2` `p1 != p2`	当 first 和 second 成员分别相等时，两个 pair 相等。相等性判断利用元素的==运算符实现

- 返回pair函数
  - 我们可以隐式的用列表初始化返回一个pair的函数

```
pair<string, int>
process(vector<string> &v)
{
    // 处理 v
```

    - 使用隐式返回

```
    if (!v.empty())
        return {v.back(), v.back().size()}; // 列表初始化
    else
        return pair<string, int>(); // 隐式构造返回值
}
```

    - 使用make-pair返回

```
if (!v.empty())
    return make_pair(v.back(), v.back().size());
```

- 关联容器概述
  - 关联容器不支持顺序容器的位置相关操作，比如pushback，pushfront
  - 关联容器不支持构造函数或插入这些接受一个元素值和一个数量值的操作
  - 关联容器的迭代器都是双向的
  - 传递给排序算法的可调用对象必须满足与关联容器中关键词一样的类型要求
  - 有序容器的关键词类型
    - 我们可以提供自己定义的操作来代替关键词上的运算符，所提供的操作必须在关键词类型上定义一个严格弱序，可以将严格弱序定义为小于等于
    - 严格弱序满足的条件
      - □ · 两个关键字不能同时“小于等于”对方；如果k1“小于等于”k2，那么k2绝不能“小于等于”k1。
      - □ · 如果k1“小于等于”k2，且k2“小于等于”k3，那么k1必须“小于等于”k3。
      - □ · 如果存在两个关键字，任何一个都不“小于等于”另一个，那么我们称这两个关键字是“等价”的。如果k1“等价于”k2，且k2“等价于”k3，那么k1必须“等价于”k3。
      - □ 如果两个关键字是等价的（即，任何一个都不“小于等于”另一个），那么容器将它们视作相等来处理。当用作map的关键字时，只能有一个元素与这两个关键字关联，我们可以用两者中任意一个来访问对应的值。

- 关联容器的类型与迭代器
  - 关联容器的类型别名
    - key--》下标，map-》值，value-》下标和值

表 11.3：关联容器额外的类型别名	
key_type	此容器类型的关键字类型
mapped_type	每个关键字关联的类型；只适用于 map
value_type	对于 set，与 key_type 相同 对于 map，为 pair<const key_type, mapped_type>

  - map-set
    - 在set类型中，key和value的type是一样的。
    - 在map中，元素是关键字-值对，即每个元素都是是一个pair对象，都包含一个关键词和一个关键的值，
    - map中元素部分是const的

```
set<string>::value_type v1;       // v1 是一个 string
set<string>::key_type v2;         // v2 是一个 string
map<string, int>::value_type v3; // v3 是一个 pair<const string, int>
map<string, int>::key_type v4;    // v4 是一个 string
map<string, int>::mapped_type v5;// v5 是一个 int
```

    - 我们可以使用作用域运算符来提取一个类型的成员——例如map《string，int》：：key-type
    - 只有map类型才定义了mapped-type
    - map是由一个一个pair构成 的，我们可以改变pair的值，但是不能改变关键词成员的值
  - 关联容器的迭代器
    - map的迭代器
      - 迭代器指向的是value-type，其first成员保存const关键词不能改变
    - set的迭代器
      - set的迭代器都是const的，可以读取但是不能更改
- 关联容器和算法
  - 我们通常不对关联容器使用泛型算法，关键词为const这一特性使他难以使用
  - 关联容器可用于只读取元素的算法，对关联容器使用算法的时候，要么是将它当作一个源序列，要么当作一个目的位置
  - 使用泛型算法和迭代器的时候，除非其只读取元素，否则是不能使用的
  - 调用自有函数的算法比泛型算法要快很多
- 关联容器的操作
  - 添加元素
    - 关联容器自带insert成员，有两个版本，分别接受一对迭代器或者一个初始化列表
    - set
      - 对于set来说，只有第一个带此关键字的元素才被插入到容器中

```
vector<int> ivec = {2,4,6,8,2,4,6,8};        // ivec 有 8 个元素
set<int> set2;                                // 空集合
set2.insert(ivec.cbegin(), ivec.cend());      // set2 有 4 个元素
set2.insert({1,3,5,7,1,3,5,7});               // set2 现在有 8 个元素
```

    - map
      - 对于一个map进行insert操作的时候，必须记住元素类型是pair

- 如果没有现成的对象，可以直接在insert的括号里插入一个pair

```
// 向 word_count 插入 word 的 4 种方法
word_count.insert({word, 1});
word_count.insert(make_pair(word, 1));
word_count.insert(pair<string, size_t>(word, 1));
word_count.insert(map<string, size_t>::value_type(word, 1));
```

- insert的操作
  - 添加单一元素的insert和emplace版本返回一个pair，告诉我们插入操作是否是成功的

表 11.4：关联容器 insert 操作

c.insert(v) c.emplace(*args*)	v 是 value_type 类型的对象；*args* 用来构造一个元素 对于 map 和 set，只有当元素的关键字不在 c 中时才插入（或构造）元素。函数返回一个 pair，包含一个迭代器，指向具有指定关键字的元素，以及一个指示插入是否成功的 bool 值。 对于 multimap 和 multiset，总会插入（或构造）给定元素，并返回一个指向新元素的迭代器
c.insert(b, e) c.insert(il)	b 和 e 是迭代器，表示一个 c::value_type 类型值的范围；il 是这种值的花括号列表。函数返回 void 对于 map 和 set，只插入关键字不在 c 中的元素。对于 multimap 和 multiset，则会插入范围中的每个元素
c.insert(p, v) c.emplace(p, *args*)	类似 insert(v)（或 emplace(*args*)），但将迭代器 p 作为一个提示，指出从哪里开始搜索新元素应该存储的位置。返回一个迭代器，指向具有给定关键字的元素

  - 对于map，set：返回一个pair，包括指向那个元素的迭代器和一个bool值，如果插入成功表示1，插入失败为0（里面已经有了）
  - 对于mutimap：只返回一个迭代器指向那个元素

- 删除元素
  - 关联容器定义了三个版本的earse
    - 前两个版本都和其他容器一样，传递一个迭代器或者一个迭代器的元素范围，之后删除
    - 还有一个版本：删除关键字元素
      - 对于map：删除成功范围1，删除失败范围0
      - 对于mutimap：删除成功返回删除数字，删除失败返回0
      - 操作

表 11.5：从关联容器删除元素

c.erase(k)	从 c 中删除每个关键字为 k 的元素。返回一个 size_type 值，指出删除的元素的数量
c.erase(p)	从 c 中删除迭代器 p 指定的元素。p 必须指向 c 中一个真实元素，不能等于 c.end()。返回一个指向 p 之后元素的迭代器，若 p 指向 c 中的尾元素，则返回 c.end()
c.erase(b, e)	删除迭代器对 b 和 e 所表示的范围中的元素。返回 e

- 查找元素
  - 关联容器自带find查找，其速度要比泛型算法的查找要快很多

表 11.7：在一个关联容器中查找元素的操作

lower_bound 和 upper_bound 不适用于无序容器。 下标和 at 操作只适用于非 const 的 map 和 unordered_map。

c.find(k)	返回一个迭代器，指向第一个关键字为 k 的元素，若 k 不在容器中，则返回尾后迭代器
c.count(k)	返回关键字等于 k 的元素的数量。对于不允许重复关键字的容器，返回值永远是 0 或 1
c.lower_bound(k)	返回一个迭代器，指向第一个关键字不小于 k 的元素
c.upper_bound(k)	返回一个迭代器，指向第一个关键字大于 k 的元素
c.equal_range(k)	返回一个迭代器 pair，表示关键字等于 k 的元素的范围。若 k 不存在，pair 的两个成员均等于 c.end()

        - mutimap、set
            - 对于muti类的map和set，bound迭代器可能指向一个具有给定关键词的元素，也可能不指向。如果关键词不在容器中，则lowerbound会返回关键词的第一个安全插入点，不影响容器中元素顺序的插入位置
            - 我们也可以使用equal-range
                - equal-range返回一个迭代器pair，一个是指向第一个与关键词匹配的元素，一个是指向最后一个匹配元素的位置
    - map容器的下标操作
        - map和unordered map容器提供了下标运算符和一个对应at函数
            - 操作
                - 表 11.6：map 和 unordered_map 的下标操作

c[k]	返回关键字为 k 的元素；如果 k 不在 c 中，添加一个关键字为 k 的元素，对其进行值初始化
c.at(k)	访问关键字为 k 的元素，带参数检查；若 k 不在 c 中，抛出一个 out_of_range 异常（参见 5.6 节，第 173 页）

        - 其他容器不支持下标的原因
            - set没有与关键词相对应的值，元素本身就是关键字
            - 我们不能对一个mutimap或者一个unordered mutimap进行下标操作，因为这些容器中可能有多个值与一个关键字进行关联
- 无序容器
    - 无需关联容器不是使用比较运算符来组织元素，而是使用一个哈希函数和关键字类型的==运算符，在关键字类型元素没有明显的序关系的情况下，无序容器是非常有用的
    - 定义在unorder map中
    - 如果关键字类型是无序的，或者性能测试发现问题可以用哈希技术解决，就可以使用无序容器
    - 无序容器还提供了和有序容器相同的操作
    - 桶操作
        - 操作

表 11.8：无序容器管理操作	
桶接口	
c.bucket_count()	正在使用的桶的数目
c.max_bucket_count()	容器能容纳的最多的桶的数量
c.bucket_size(n)	第 n 个桶中有多少个元素
c.bucket(k)	关键字为 k 的元素在哪个桶中
桶迭代	
local_iterator	可以用来访问桶中元素的迭代器类型
const_local_iterator	桶迭代器的 const 版本
c.begin(n), c.end(n)	桶 n 的首元素迭代器和尾后迭代器
c.cbegin(n), c.cend(n)	与前两个函数类似，但返回 const_local_iterator
哈希策略	
c.load_factor()	每个桶的平均元素数量，返回 float 值
c.max_load_factor()	c 试图维护的平均桶大小，返回 float 值。c 会在需要时添加新的桶，以使得 load_factor<=max_load_ factor
c.rehash(n)	重组存储，使得 bucket_count>=n 且 bucket_count>size/max_load_factor
c.reserve(n)	重组存储，使得 c 可以保存 n 个元素且不必 rehash

- 提供hash模板

```
size_t hasher(const Sales_data &sd)
{
    return hash<string>()(sd.isbn());
}
bool eqOp(const Sales_data &lhs, const Sales_data &rhs)
{
    return lhs.isbn() == rhs.isbn();
}

using SD_multiset = unordered_multiset<Sales_data,
                    decltype(hasher)*, decltype(eqOp)*>;
// 参数是桶大小、哈希函数指针和相等性判断运算符指针
SD_multiset bookstore(42, hasher, eqOp);

// 使用 FooHash 生成哈希值；Foo 必须有==运算符
unordered_set<Foo, decltype(FooHash)*> fooSet(10, FooHash);
```